E-ADTs: Turbo-Charging Complex Data

Praveen Seshadri
Computer Science Department
Cornell University, Ithaca, NY
praveen@cs.cornell.edu

Miron Livny
Computer Sciences Department
U.Wisconsin, Madison WI
miron@cs.wisc.edu

Raghu Ramakrishnan
Computer Sciences Department
U.Wisconsin, Madison WI
raghu@cs.wisc.edu

Abstract

The next generation of database applications will be dominated by rich and complex data types. The ADT technology of today's object-relational database systems cannot provide adequate performance for such applications. The basic stumbling block is the lack of semantic knowledge provided to the database system about the ADT operations. We are developing novel "Enhanced ADT" (E-ADT) technology that overcomes this problem, while retaining the extensibility of the database system. In effect, this "turbo-charges" the complex data types, resulting in dramatically improved performance. E-ADT technology is being demonstrated on several kinds of multi-media data in the $\mathcal{PREDATOR}$ database system at Cornell.

1 Introduction

We are witnessing an explosion in the volume and complexity of digital information that people want to access and analyze. Much of this data is "multi-media" like images, video, audio, and documents. Several other kinds of complex data are also important, including geographical objects, chemical and biological structures, mathematical entities like matrices and equations, and financial data like time-series. Database systems must efficiently support queries over such richly structured data; otherwise, they will fast become "roadkill on the information super-highway" [?]. To meet this challenge, novel "Enhanced ADT" (E-ADT) technology is being developed in the $\mathcal{PREDATOR}$ database system at Cornell.

The existing support for complex data in object-relational database systems (OR-DBMSs) is based on Abstract Data Types (ADTs), which were adapted from programming language concepts [?] to databases in the 1980s [?, ?]. Prominent OR-DBMSs based on ADTs include Illustra [?], Postgres [?] and Paradise [?]. The OR-DBMS maintains a table of ADTs and new ADTs may be added by a database developer or user. Each ADT provides a number of primitive operations for manipulating or querying values of the type. An ADT for images might provide operations *Rotate(I, Angle), Clip(I, Region)*, and *Overlay(I1, I2)*, which are composed to form query expressions like *Overlay(Rotate(I1, 90), Clip(I2, {0, 0, 100, 200}))* that can be embedded within an SQL query. This is the current state of the art, which we call the "ADT approach".

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

2 Motivation

Operations on complex ADTs can be expensive (for example, *Rotate(Image)*); indeed, the cost of ADT operations often dominates the overall execution cost of a query. Obviously, query processing and optimization should attempt to reduce the cost of ADT operations. There are two issues to consider: (a) What are possible optimizations on such queries? (b) How can these optimizations be systematically applied in a DBMS?

2.1 Possible Optimizations

Consider an OR-DBMS based on the ADT approach, and assume that an image ADT has been added. While image data is stored on disk in a compressed format, the ADT operations are implemented on an uncompressed main-memory image data structure. Consequently, the image argument of any operation is converted to its main-memory uncompressed form before an operation is invoked on it.

An earth scientist maintains a table of geographic data, each entry having a satellite photograph and several other columns. An SQL query asks for a rotated portion of each photograph of the arctic region. The cost of this query is dominated by the operations on the images.

```
SELECT Clip(Rotate(G.Photo, 90), {0, 0, 100, 200})
FROM  GeoData G
WHERE G.Region = 'arctic'
```

How is this query evaluated in current OR-DBMSs[1]? For every data entry corresponding to the arctic region, the Photo attribute is retrieved from disk and decompressed into a main-memory image. The *Rotate* operation is then applied to it, and the resulting image is written to a compressed disk-resident form. The *Clip* operation is then applied with the intermediate result image as its input. This input image is decompressed to a main-memory form, it is clipped to the desired dimensions, and the resulting image is written out to disk. One could improve this execution strategy as follows:

- It is unnecessary for *Rotate* to compress and write its result to disk. Instead, it could be passed directly in memory to *Clip*.
- *Rotate* is an expensive operation, whose cost depends on the size of the image being rotated. It would be cheaper to evaluate the equivalent expression *Rotate(Clip(G.Photo, {0, 0, 200, 100}), 90)*. By performing *Clip* early, there can be significant reductions in the cost of *Rotate*. Note that the arguments of *Clip* have changed as a result of its being applied before *Rotate*.
- If *Clip* is applied before *Rotate*, the entire image does not need to be retrieved from disk. Only the appropriate portion of the image is needed.

A combination of these strategies can lead to performance improvements of an order of magnitude. Essentially, an entire ADT expression needs to be treated *declaratively* and optimized. The textual representation of an expression should not specify an evaluation plan. Current OR-DBMSs based on the ADT approach violate this principle; they do not perform such optimizations. Consequently, their performance is poor.

2.2 Categories of Optimizations

Similar improvements should be applied to ADT expressions involving other complex data types like audio, time-series, matrices, etc. We can classify the optimizations into four broad categories:

- *Algorithmic:* Using different algorithms for each operation depending on the data characteristics. For example, the best algorithm to use for the *Multiply* operation on two matrices depends on the sizes of the matrices and the amount of memory available.

[1] Our presentation is of the execution of a "typical" OR-DBMS like Illustra. Systems like Paradise [?] incorporate some of the optimizations we suggest.

- *Transformational:* Changing the order of operations. The motivating example shows how *Clip* can be applied before *Rotate*.
- *Constraint:* Pushing constraints through the expression. The constraints may involve selecting a portion of the data, specifying a certain data resolution, or requiring a particular physical property. Consider the expression *Clip(Smoothen(I), Region)* on images. The whole image does not need to be smoothened; only a region of it is needed.
- *Pipelining:* Pipelined execution of operations to avoid materializing intermediate results. This is crucial for large data types like audio and video. It may not be possible to fit an entire uncompressed audio (or video) object in memory. The only reasonable way to access the data is to treat it as a stream. When a sequence of operations has to be applied to such a large object (for example, *IncrTreble(DecrVolume(Audio))*), pipelining the operations is clearly better than generating entire intermediate results for each operation. This is similar to the use of lazy evaluation [?] in functional programming.

3 A Performance Demonstration

What is the effect of such optimizations? We answer this with an initial performance demonstration in $\mathcal{PREDATOR}$. We use a simple data set containing 74 images of cars, compressed using JPEG. A Cars relation is created, with a name assigned to each image. Each image column value contains an Id for the image and its bounding box information. The size of the compressed images ranges from 23K to 266K, with an average of 65K. This is a relatively small amount of data, but it serves to demonstrate the issues involved. The average size of the uncompressed images in memory is 0.8MB representing more than a 10-fold increase from the size on disk. Standard JPEG libraries are used to perform compression and decompression. Experiments were run on a Sparc20 machine with 64MB of physical memory, of which 8 MB was used as a database buffer pool. The buffer space was sufficient to hold the compressed data, and each individual uncompressed image easily fit in memory. The system is CPU-bound in all these experiments, with the expensive image operations using most of the execution time.

Experiment 1: The first experiment examines the effects of the Pipelining optimizations. We noted in Section **??** that in the ADT approach, each operation reads (and decompresses) its input from a disk-based representation, and writes (and compresses) its output. We call this strategy DISK. An improvement is to pass intermediate results in their main-memory form. We call this strategy MEM. Finally, if there is a sequence of operations, we could pipeline their execution by establishing an image row iterator. We call this strategy PIPE. We expect PIPE to be significantly better than MEM only when the intermediate results are larger than main-memory; that is not the case for these images.

```
SELECT Height(Negative(C.picture))
FROM Cars C;
```

In the query above, *Negative()*, which inverts the pixel values, is a relatively cheap operation. For all three strategies, *C.picture* must be read and decompressed. The DISK strategy compresses and writes the result out, whereas the MEM strategy does not. The *Height* operation is applied so that the display time for the result does not distort the measurements. The height is obtained from the image bounding box, so the actual image is not required. We gradually vary the query by introducing additional invocations of the *Negative()* operation. For instance, *SELECT Height(Negative(Negative(C.picture)))* requires one additional compression and decompression using the DISK strategy. The results are shown in Figure **??**. Along the X-axis is the number of *Negative()* operations in the SELECT clause. The Y-axis shows the execution time.

As the number of operations in the image expression increases, the effect of the compression and decompression at the image boundaries dominates. Consequently, the MEM strategy which avoids these unnecessary costs is significantly cheaper than DISK. As expected, the PIPE strategy does only marginally better than MEM, since the intermediate results fit easily in memory. The MEM strategy, which is easier to implement, has been used as the default for the remaining experiments.

Experiment 2: The second experiment examines the effects of Order and Constraint optimiza-

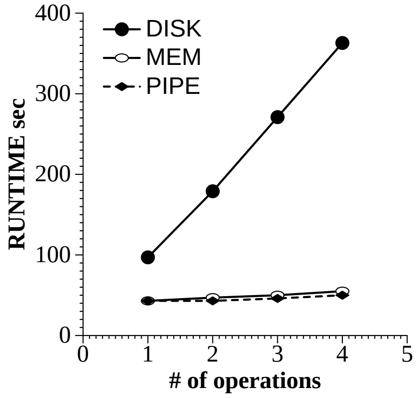

Figure 1: Pipelining

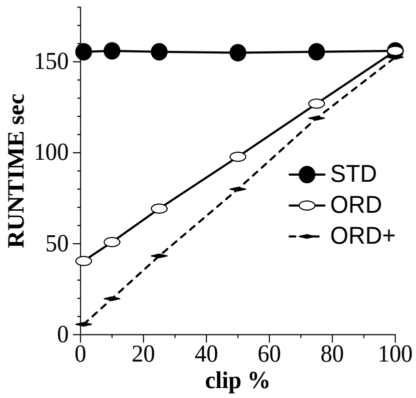

Figure 2: Varying Clip Size

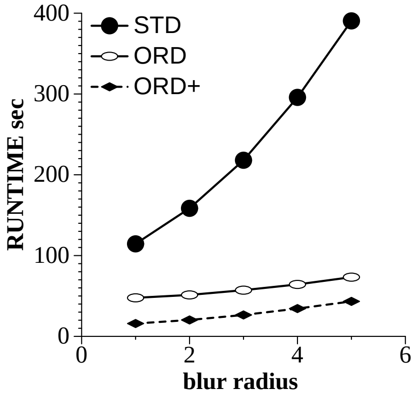

Figure 3: Varying Blur Radius

tions. The following query is used:

```
SELECT Height(Clip(Blur(C.picture, <radius>), <region>))
FROM Cars C;
```

The result of a *Blur* operation is an image in which every pixel's value is the average of the pixels within *radius* of it in the input image. *Blur* is an expensive operation, especially when *radius* is large. In the standard ADT approach, the images will first be blurred, then clipped to the appropriate region. We call this the STD strategy. Using the Order optimizations, the *Clip* operation could be performed before the *Blur* operation. We call this the ORD strategy. Finally, if *Clip* is being performed first, it can use the region to constrain the retrieval and decompression of the images. We call this the ORD+ strategy.

Figure **??** shows how these strategies perform when the *Blur radius* is fixed at 2 and the size of the Clip region is varied as a percentage of the image size. Since blurring an image is expensive, both ORD and ORD+ perform very much better than STD when a small region needs to be clipped. Further, it is evident that ORD+ is also significantly better than ORD because it requires a smaller portion of each image to be retrieved and decompressed. Figure **??** fixes the *Clip region* at 10% of the image size, and varies the *Blur radius* from 1 to 5. As the *radius* increases, the cost of the blurring grows quadratically. For the STD strategy, since this is the dominant cost, the overall execution time also grows quadratically. In comparison, in ORD and ORD+, the clips are performed early. The difference between ORD and ORD+ is the difference in the retrieval costs; it is evident that the execution time of ORD is dominated by the retrieval and decompression time. Consequently, the quadratic change in the *Blur* cost is not apparent.

Experiment 3: The third experiment demonstrates a very simple Constraint optimization so as to emphasize that constraints do not necessarily have to derive from a *Clip* operation. Consider the query below:

```
SELECT Area(Overlap(C1.picture, C2.picture))
FROM Cars C1, Cars C2
```

```
WHERE C2.name = "Alfa_Romeo_8C_2300"
```

The query needs to find the area of overlap between pairs of images. There is really no need to compute the actual overlap of the images; their bounding boxes contain the necessary information to find the area of overlap. If this constraint is pushed into the computation of the *Overlap* operation, the performance is greatly improved. Measurements show that the execution time goes from 79.5 seconds to 0.58 seconds!

4 A Systematic Optimization Framework

Each of the demonstrated optimizations can result in an order of magnitude in performance improvements. Their combination can result in even greater improvements. We have shown similar improvements for queries involving time-series data [?]. However, it is not sufficient to observe that these optimizations are possible. There must also be an architectural framework to apply them in a correct and cost-based manner, so that the efficiency of a query is improved. At the same time, the type system should remain extensible, so that new types can be added incrementally. Continuing with the example using images, we note that:

- The best algorithms used for each operation can depend on the size of the input image, the amount of memory available, the storage format of the image and the values of arguments to the operation. Consequently, for any complex expression involving multiple image operations, there are several possible evaluation plans. Deciding between different plans requires cost-based optimization of the image expression.
- The image expression may be evaluated several times during the course of the query. Its evaluation plan should be chosen before the query starts executing, because it is unreasonable to explore the large space of options at runtime. Further, the cost of the chosen plan is used at compile time by the SQL optimizer to "place" the evaluation of the expression at the appropriate position in the join evaluation tree [?, ?].
- In order to have meaningful compile-time optimization, collective meta-information like the storage formats and size statistics need to be maintained over all the pertinent images (in this case, over all photographs in the GeoData table). The meta-information maintained and the optimizations to be applied are specific to each ADT.

Current OR-DBMSs lack such an optimization infrastructure. The E-ADT paradigm *does* provide the framework to support such optimizations. It "enhances" traditional ADTs so that combinations of ADT operations are treated as declarative expressions that are optimized in a cost-based manner. At the same time, the modularity of the type system is retained.

4.1 Enhanced Abstract Data Types

An Enhanced Abstract Data Type (E-ADT) is an enhancement of the concept of ADTs in database systems to improve the performance of query processing. In addition to providing the standard ADT functionality, each E-ADT may support one or more of the following enhancements through a uniform internal interface:

- *Query Language:* An E-ADT can provide parsing capabilities for a native query language with which expressions over values of that E-ADT can be specified. As an important special case, the language provided can use the simple functional syntax found in today's OR-DBMSs – this means that there is no need to change SQL syntax on account of E-ADTs.

- *Query Operators and Optimization:* An E-ADT can provide optimization routines that will translate a language expression into a query evaluation plan in its own evaluation algebra.

- *Query Evaluation:* An E-ADT can provide routines to execute the optimized plan.

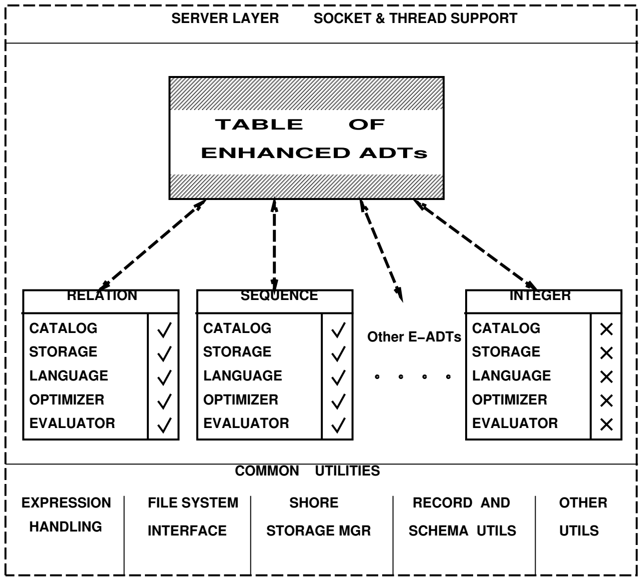

Figure 4: $\mathcal{PREDATOR}$ System Architecture

- *Catalog Management:* An E-ADT can provide catalog routines so that schema information can be stored and statistics maintained on values of that E-ADT .

- *Storage Management:* An E-ADT can provide multiple physical implementations of values of its type. Many existing OR-DBMSs already support this feature.

4.2 The $\mathcal{PREDATOR}$ DBMS

$\mathcal{PREDATOR}$[2] is a client-server OR-DBMS in which the server is a loosely-coupled system of E-ADTs. The core of the system is an extensible table in which E-ADTs are registered. The server is built on top of a layer of common database utilities that all E-ADTs can use. An important component of the utility layer is the SHORE Storage Manager [?] library, which provides facilities for persistent storage, concurrency control, recovery and buffer management. The high-level picture of the system is shown in Figure ??.

Some of the basic types like integers do not support any enhancements. The figure shows two E-ADTs that do support enhancements: relations and sequences (i.e. time-series). An image E-ADT has also been added to the system. A complex object like an image or a time-series can be a field within a relational tuple. An important feature of the system design is that E-ADTs are modular, with all E-ADTs presenting an identical internal interface. This serves a dual purpose:

- The development of optimizations for each E-ADT proceeds independently. The extensibility of the system would be compromised if each E-ADT had to be aware of all the other E-ADTs in the system.
- The interaction between the E-ADTs (especially between the SQL query processing engine and the individual E-ADT expressions) happens through the uniform internal interface, independent of the implementation of each E-ADT.

Due to space constraints, we do not discuss the detailed optimizations on the operations of each data type. More importantly, we only present a brief summary of the interaction mechanism between data types. A query is broken into components that correspond to sub-expressions of each E-ADT. Each sub-expression is treated as a declarative query and is parsed, optimized, and executed by its

[2] $\mathcal{PREDATOR}$ stands for the PRedator Enhanced DAta Type Object Relational DBMS

own E-ADT. It is difficult to build a DBMS with an extensible type system, while at the same time providing this rich functionality in each E-ADT. In fact, the E-ADT paradigm affects code design at many levels of the query processing engine. The details are in a longer version of this paper [?].

5 Related Work

This section is a brief survey of the related work in this area. [?] presents a more exhaustive description of related research and describes several exciting areas for future research enabled by the E-ADT paradigm.

Postgres and Illustra: The issues regarding support for ADTs in database systems were first explored in [?] and [?]. This led to the Postgres research DBMS [?] and its commercial version, Illustra [?]. The Postgres project explored issues dealing with the storage and indexed retrieval of ADTs. It also stressed that functions associated with ADTs could be expensive [?], and that special relational optimization techniques are necessary when such functions are present [?]. The "ADT approach" described in this paper corresponds closely to Illustra's technology. Currently, Illustra is being integrated into the Informix Universal Server [?] and several modifications are being made to improve the parallel evaluation of ADT expressions [?].

Paradise: The Paradise client-server DBMS [?] is developing ADT extensions for the parallel execution of operations on spatial, geographic, and scientific data. This work concentrates on issues of scalable parallelism and the use of tertiary storage for large ADTs. The importance of parallelism for ADTs arises from the large size of complex data types, and the high cost of operations on them. From the E-ADT viewpoint, parallelism is yet another possible benefit of declarative ADT expressions. Consequently, the Paradise work on parallel execution of ADT methods fits in neatly with the E-ADT paradigm. Paradise also implements some optimizations that allow ADT operations to retain intermediate results in memory.

There is a large amount of research in areas like object-oriented query processing [?, ?] that is related to our work. For example, Aberer and Fischer [?] observed that the semantic information about methods in an OO-DBMS can be used to optimize the query. Much of the recent work on extensible query optimization (like [?]) is also relevant. Several researchers have been exploring database support for collection types [?, ?] and scientific data [?, ?]. There has been an increasing interest in "special-purpose" database systems, which focus on data types or application domains of interest. This includes work on supporting queries over arrays, financial data, images, video, text, music, source code, etc. Our research impacts all this work, because our technology enables the domain specific query languages and optimizations developed in all these research projects to be integrated within a common database system. Finally, there are many similarities between the support for multiple data types, and the support of heterogeneous database systems. The loosely-coupled approach that we have taken encourages this comparison, and also opens up the possibility of building heterogeneous E-ADTs on top of the OLE-DB standard [?].

6 Conclusion

The E-ADT paradigm is a novel approach to database systems design that "turbo-charges" the use of complex data types. E-ADTs are being built in the $\mathcal{PREDATOR}$ database system, and initial results indicate an order of magnitude increase in performance. Consequently, we believe that the next-generation of object-relational database systems should be based on E-ADTs.

Acknowledgments

Praveen Seshadri was supported by IBM Research Grant 93-F153900-000 by an IBM Cooperative Fellowship, and by NASA Research Grant NAGW-3921. Miron Livny and Raghu Ramakrishnan were supported by NASA Research Grant NAGW-3921. Raghu Ramakrishnan was also supported by a Packard Foundation Fellowship in Science and Engineering, a Presidential Young Investigator Award with matching grants from DEC, Tandem and Xerox, and NSF grant IRI-9011563.

References

[AF95] Karl Aberer and Gisela Fischer. Semantic query optimization for methods in object oriented database systems. In *Proceedings of the Eleventh IEEE Conference on Data Engineering, Taipei, Taiwan*, pages 70–79, 1995.

[Bla96] Jose Blakeley. Data Access for the Masses through OLE-DB. In *Proceedings of ACM SIGMOD '96 International Conference on Management of Data, Montreal, Canada*, pages 161–172, 1996.

[Cat94] R.G.G. Cattell. *The Object Database Standard:ODMB-93.* Morgan-Kaufman, 1994.

[CD92] S. Cluet and C. Delobel. A General Framework for the Optimization of Object-Oriented Queries. In *sigmod92*, pages 383–392, 1992.

[CDF+94] M.J. Carey, D.J. DeWitt, M.J. Franklin, N.E. Hall, M. McAuliffe, J.F. Naughton, D.T. Schuh, M.H. Solomon, C.K. Tan, O. Tsatalos, S. White, and M.J. Zwilling. Shoring up persistent objects. In *Proceedings of ACM SIGMOD '94 International Conference on Management of Data, Minneapolis, MN*, pages 526–541, 1994.

[Che96] Mitch Cherniak. Form(ers) Over Function(s): The KOLA Reference Manual . Unpublished draft, Brown University, 1996.

[CS96] Surajit Chaudhuri and Kyuseok Shim. Optimization of queries with user-defined predicates. In *Proceedings of the Twenty Second International Conference on Very Large Databases (VLDB), Bombay, India*, pages 87–98, September 1996.

[DeW95] David J. DeWitt. DBMS: Roadkill on the Information Superhighway? Invited Talk: VLDB 95, 1995.

[DKL+94] D.J. DeWitt, N. Kabra, J. Luo, J.M. Patel, and J. Yu. Client-Server Paradise. In *Proceedings of the Twentieth International Conference on Very Large Databases (VLDB), Santiago, Chile*, September 1994.

[Hel95] Joseph M. Hellerstein. *Optimization and Execution Techniques for Queries With Expensive Methods.* PhD thesis, University of Wisconsin, August 1995.

[Ill94] Illustra Information Technologies, Inc, 1111 Broadway, Suite 2000, Oakland, CA 94607. *Illustra User's Guide*, June 1994.

[Inf96] Informix Software, Inc. *Informix and Illustra Merge to Form Universal Server.* Informix White Paper, http://www.informix.com/informix/corpinfo/zines/whitpprs/illustra/ifxillus.htm, February 1996.

[Jhi88] Anant Jhingran. A Performance Study of Query Optimization Algorithms on a Database System Supporting Procedures. In *Proceedings of the Fourteenth International Conference on Very Large Databases*, pages 88–99, 1988.

[Jon87] S.L. Peyton Jones. *The Implementation of Functional Programming Languages.* Prentice Hall, 1987.

[LZ74] B. Liskov and S. Zilles. Programming with Abstract Data Types. In *SIGPLAN Notices*, April 1974.

[MDZ93] Gail Mitchell, Umeshwar Dayal, and Stanley Zdonik. Control of an Extensible Query Optimizer: A Planning-Based Approach. In *Proceedings of the Nineteenth International Conference on Very Large Databases (VLDB), Dublin, Ireland*, pages 517–528, 1993.

[MV93] D. Maier and B. Vance. A call to order. In *Proceedings of the Twelfth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, Washington, DC*, 1993.

[Ols96] Mike Olson, 1996. Personal Communication.

[SLR96a] Praveen Seshadri, Miron Livny, and Raghu Ramakrishnan. The Case for Enhanced Abstract Data Types. Submitted for Publication, 1996.

[SLR96b] Praveen Seshadri, Miron Livny, and Raghu Ramakrishnan. The Design and Implementation of a Sequence Database System. In *Proceedings of the Twenty Second International Conference on Very Large Databases (VLDB), Bombay, India*, pages 99–110, September 1996.

[SRG83] M. Stonebraker, B. Rubenstein, and A. Guttman. Application of Abstract Data Types and Abstract Indices to CAD Data Bases. In *Proceedings of the Engineering Applications Stream of Database Week*, San Jose, CA, May 1983.

[SRH90] Michael Stonebraker, Lawrence Rowe, and Michael Hirohama. The Implementation of POSTGRES. *IEEE Transactions on Knowledge and Data Engineering*, 2(1):125–142, March 1990.

[Sto86] Michael Stonebraker. Inclusion of New Types in Relational Data Base Systems. In *Proceedings of the Second IEEE Conference on Data Engineering*, pages 262–269, 1986.

[WG93] Richard Wolniewicz and Goetz Graefe. Algebraic Optimization of Computations over Scientific Databases. In *Proceedings of the Nineteenth International Conference on Very Large Databases (VLDB), Dublin, Ireland*, pages 13–24, 1993.

[Won94] Limsoon Wong. *Querying Nested Collections.* PhD thesis, U.Pennsylvania, 1994.